SABRE: A Text-to-Database Search Architecture for Relational Databases

Mikael Sales[1], Marlon Gonçalves Duarte[2], Wellington Franco[3], Angelo Brayner[4]

[1]Kunumi Lab – Universidade Federal do Ceará (UFC), Fortaleza – CE

[2]Departamento de Computação – Universidade Federal do Ceará (UFC)

mikaelmfsales@gmail.com, marlongduarte@alu.ufc.br, wellington@crateus.ufc.br, brayner@dc.ufc.br

Abstract. *Relational database queries depend on SQL and prior knowledge of the schema. This work proposes SABRE, a plain-text search architecture for relational databases based on the vectorization of tuples and queries and on retrieval by semantic similarity. The evaluation used a pilot study for selecting the embedding model and experiments with the TPC-H benchmark, comparing the relational and vector modes in terms of retrieval quality, latency, and end-to-end cost. The vector approach achieved a Recall@K of 0.8459, a Precision@K of 0.8039, and an NDCG@K of 0.8886, indicating good coverage of relevant tuples and adequate ranking quality. However, SQL execution showed lower latency in 21 of the 22 queries, and the vector cost was dominated by the database search stage. The results indicate that SABRE is suitable as a complementary mechanism for semantic retrieval and candidate generation, but not as a direct substitute for exact SQL execution in analytical workloads such as TPC-H.*

1. Introduction

Traditionally, queries in relational databases are performed through SQL commands, requiring the user to know the schema, the tables involved, and the predicates necessary to correctly formulate the desired query [Mama and Machkour 2025]. In parallel, Large Language Models (LLMs) have been applied in several areas, such as text generation, translation, summarization, and information retrieval, often with the support of *embeddings* to semantically represent texts, queries, and documents [Kumar 2024]. In this context, vector representation through *embeddings* has been consolidated as a practical approach to model semantic similarity between texts and, by extension, between intent descriptions and data objects [Garg et al. 2018]. Formally, according to [Corsi 2025], an *embedding* is a dense floating-point vector of fixed dimensionality, learned in such a way as to preserve semantic relationships of the original data in a continuous space, in which geometric proximity reflects similarity of meaning. From classical distributed representations to pretrained contextual models, *embeddings* enable high-dimensional similarity comparisons and support large-scale semantic retrieval applications [Greiner-Petter et al. 2020, Chen et al. 2024b].

For this comparison to be feasible in real scenarios, vectors need to be stored, indexed, and queried efficiently, since the search cost grows with the volume of data and with the dimensionality of the vector space [Douze et al. 2025]. In this context, vector databases emerged to offer specialized index structures, such as Hierarchical Navigable Small World - HNSW [1] and Inverted File Index - IVF [2], in addition to native similarity op-

[1] MongoDB [2] Apache

erators for *top-k* queries, persistence mechanisms, and resources for updating and managing the vector lifecycle [Pan et al. 2024b]. These characteristics have made vector search a relevant alternative for applications in which exact matching by equality or logical predicate is not sufficient to express the intent of the query [Mathur and Chhabra 2024].

In parallel, relational Database Management Systems - DBMSs remain the main infrastructure for structured data and analytical workloads, due to the maturity of their optimizers and the efficient and deterministic execution of operations such as selections, joins and aggregations [Abouzeid et al. 2009, Freitag et al. 2020]. Consolidated benchmarks, such as TPC-H [3], provide a workload to evaluate the processing in decision support queries, covering scenarios with multiple tables, filters, groupings and aggregations [He et al.]. In this sense, TPC-H constitutes a suitable basis for comparing approaches with distinct semantics under the same set of queries and data [Yong and Teh 2025].

In addition to semantically expanding access to relational data, it is necessary to evaluate these approaches in terms of latency and end-to-end cost, since response time is an important metric in database applications [Petrola et al. 2025]. Despite the interest in the integration of search mechanisms, it is still not sufficiently clear, in a measurable and reproducible way, how similarity-based retrieval behaves when confronted with traditional relational execution over the same analytical workload [Pan et al. 2024a]. According to [Chen et al. 2024a], *embeddings* expand the class of questions that can be asked about relational data, but introduce an approximate, model-dependent semantics, distinct from the exact semantics of SQL. In addition, part of the available evaluations emphasizes only the latency of the Approximate Nearest Neighbor - ANN operator, leaving in the background relevant costs of the complete process, such as the generation of the query vector, the search over the vector corpus and the fidelity of the results in relation to the ground truth produced by relational execution [Riccardo Cappuzzo 2020, Gollapudi et al. 2023].

Given this context, this work proposes an architecture for plain-text search over relational databases, based on transforming natural language into vector representations and performing semantic similarity retrieval over vectorized tuples. Its central contribution is the definition of this architecture as a mechanism for semantic access to structured data, extending query capabilities beyond strict SQL semantics. The proposal is validated in the TPC-H domain by comparing traditional relational execution with vector-based execution using *embeddings*, considering latency, end-to-end cost, and result adherence to relational ground truth. Thus, the work combines an architectural contribution with an experimental analysis that characterizes the behavior, costs, and limitations of the approach.

The work is organized as follows. Section 2 discusses the related work. Section 3 presents the methodology of the proposed architecture. Section 4 brings together the results and their discussion, considering fidelity, latency, and *end-to-end* cost. Finally, Section 5 presents the conclusions and future perspectives.

2. Related Work

The advancement of *embeddings* and similarity search techniques has motivated the incorporation of semantic capabilities into relational DBMSs and analytical engines [Khalifeh et al. 2025]. In general terms, the related literature can be organized into

[3] TPC®

four main and technically complementary strands: (i) semantic queries over relational databases via *embeddings*, (ii) similarity-based analytical (OLAP) extensions, (iii) acceleration of vector search within PostgreSQL through architectural changes, and (iv) hybrid engines that combine structured and unstructured processing in a single query execution.

[Bordawekar and Shmueli 2017] propose enabling semantic queries in relational databases through *word embeddings*, introducing the class of *Cognitive Intelligence queries*. The approach transforms relations into a textual *corpus* (rows as sentences and the table as a document), trains a *word2vec* model ([Church 2017]) from the database itself and/or from external sources, and persists the vectors in specific internal structures of the database management system itself. Query execution is enabled by UDFs (*User-Defined Functions*) that compute similarity and allow operations such as searching for semantically close items, analogies, and inference based on learned knowledge. Although it demonstrates expressiveness, the work is predominantly conceptual and does not establish an experimental protocol, which limits its use as a *baseline* for latency comparison.

In an analytical context, the work [de Mendonça et al. 2024] discusses similarity-based OLAP queries in relational DBMSs, replacing equality-based grouping with a distance-defined *GROUP BY*. The authors implement, in PostgreSQL, *UDFs* such as `sim_metr(atrs, dist)` to assign group labels to tuples based on a threshold, extending the mechanism to metric, spatial, and temporal dimensions. The analytical granularity is controlled by varying the threshold, allowing *roll-up* and *drill-down* by re-execution. The results validate the feasibility and expressiveness of similarity-based grouping; however, the focus lies on modeling and functionality rather than on a systematic performance *benchmark* against approximate vector indexes.

In the context of vector search in PostgreSQL, PostgreSQL-V [Liu et al. 2026] addresses the performance *gap* between extensions coupled to the Postgres core and specialized libraries/engines. The proposal decouples the vector index from the traditional page architecture and *buffer manager*, preserving the DBMS interface and delegating index construction/querying to native libraries (e.g., Faiss[4], hnswlib[5], DiskANN[6]). To support updates, it adopts principles inspired by LSM (immutable segments, *flush*, and deletion marking) and mechanisms for consistency and recovery. Experimentally, the work reports expressive gains in memory (IVF_FLAT and HNSW) and on disk (DiskANN), approaching the performance of native implementations and quantifying a residual overhead associated with the database *stack* (SQL interpretation/execution and access to the *heap*). This line is directly related because it discusses, with experimental evidence, how architectural decisions impact latency and end-to-end cost in similarity queries.

Finally, AnalyticDB-V [Wei et al. 2020] explores a hybrid analytical engine for *query fusion* between structured and unstructured data, aiming to reduce the typical latency of two-stage solutions (vector search followed by filtering/analytics in a second engine). The system combines SQL execution components with ANNS modules (in C via JNI), employs indexes based on techniques such as IVFPQ, and introduces: a design in *lambda framework* to reconcile real-time ingestion with periodic index reorganization and an *accuracy-aware cost-based* optimizer, which chooses plans and hyperparameters considering selectivity and *recall*. The experiments vary selectivity by structured predicates and compare integrated execution versus *2-step*, showing substantial latency reductions

[4] Faiss [5] Hnswlib [6] DiskANN

and good throughput sustainment under mixed read/write load. This strand is relevant because it shows that the benefit of vector search depends on integration with relational processing and on operator ordering, especially in hybrid queries.

In summary, the works cover everything from the semantic expansion of relational queries via *embeddings* [Bordawekar and Shmueli 2017], through similarity-oriented OLAP [de Mendonça et al. 2024], to optimizations for efficient vector search in PostgreSQL [Liu et al. 2026] and hybrid engines with joint optimization [Wei et al. 2020]. Unlike these proposals, the present work proposes an architecture for plain-text search over relational databases, based on the transformation of natural language queries into vector representations and on similarity-based retrieval over previously vectorized tuples.

3. Methodology

This work proposes and evaluates a text-to-database search architecture for querying databases through natural language. In this architecture, the relational database is maintained as ground truth and, in parallel, converted into a vector corpus through tuple vectorization. Queries are represented in plain text, vectorized, and used for similarity-based retrieval in the vector space. As summarized in Figure 1, the study comprises a pilot study for embedding model selection and pipeline validation, followed by the comparative execution of relational and vector modes on the TPC-H benchmark, with the objective of evaluating, under a protocol, the trade-off between performance, end-to-end cost, and the fidelity of vector results in relation to the relational mode.

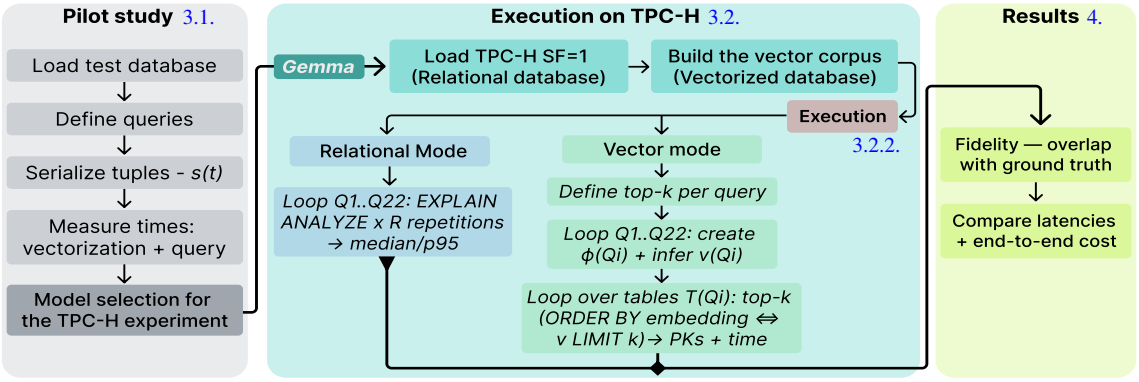

Figure 1. Methodological flowchart divided into the pilot study stage, for *pipeline* validation and *embedding* model selection; execution on TPC-H, with construction of the relational database and vector corpus; and results analysis, fidelity, latency, and *end-to-end* cost.

3.1. Pilot study

Before the experiments with TPC-H, a pilot study was conducted on a smaller relational database in order to validate operational aspects of the pipeline, such as serialization, vector materialization, instrumentation, and metric collection. The database used was *Library-System-Management-SQL* [7]. For this stage, 20 queries were defined, including 10 simple ones, without aggregations, *joins*, or filters, and 10 complex ones, containing these operations. Each query was specified in two versions: a relational one, in SQL, and another in natural language, used in vector search over the vectorized tables.

[7] https://github.com/zakafalak/Library-System-Management-SQL.git.

Vectorization in the pilot study was performed at the tuple level. Each tuple $t \in T$ was converted into a string $s(t)$ by concatenating the values of the columns returned by the DBMS, separated by line breaks. `NULL` values were mapped to the empty string. Since the serialization does not include table/column names, it is *schema-dependent*: it remains deterministic as long as the schema and the projection, column order, are not changed. A typical instance assumes the following format:

$$s(t) = \texttt{"val}_1\texttt{\textbackslash nval}_2\texttt{\textbackslash n} \ldots \texttt{\textbackslash nval}_m\texttt{"}.$$

Three *embedding* models were evaluated, as shown in Table 1, selected based on being open source, computational cost, efficiency, and dimensionality reported in the Hugging Face *leaderboard*[8]. In each vector scenario, the same model was used both for tuple vectorization and for query vectorization.

Table 1. Evaluated *embedding* models and their respective origins, sizes, and dimensionalities. The selection considered openness, cost, and efficiency, and each model was employed in the vectorization of tuples and queries in its respective experimental scenario.

Model	Source	Size	Dimension
`jina-embeddings-v2-base-en` [9]	Jina AI	137M	512
`embeddinggemma-300m` [10]	Google DeepMind	307M	768
`Qwen3-Embedding-0.6B` [11]	Alibaba Group	595M	1024

For each table T and model M, $e_M(t) = \mathit{Embed}_M(s(t))$ was computed and a vector table was materialized containing the primary key of the record and the `vector(d_M)` column via *pgvector* [12]. The vector tables were separated by model to avoid dimensionality incompatibility between the vectors of each model. Vectorization times were measured per row, per table, and in total, as well as the query response time. For the collection of response times, the executions were repeated R times, with $R = 20$, in relational and vector modes, and the median was reported as the latency metric. These measurements aimed to validate the pipeline before the main experiment.

As a result of the pilot study, a significant difference was observed in tuple vectorization time across the models, as shown in Figure 2. The Jina model (`jina-embeddings-v2-base-en`) presented the lowest vectorization time, at 0.013328ms, followed by Gemma (`embeddinggemma-300m`) with 0.052503ms. The Qwen model (`Qwen3-Embedding-0.6B`) was the most costly, at 0.546158ms, indicating greater computational overhead in the construction of the vector corpus.

For the response time of the pilot study queries, Figure 3 presents the times per query, q1–q10, for the simple queries, showing the intra-query variation among the models on a logarithmic scale. It is observed that the relational baseline and the Gemma model maintain, in all queries, latencies in the sub-millisecond to $\sim$1 ms range, with moderate variation and comparatively more stable temporal behavior across executions. The Jina model shows similar behavior in several queries, but with occasional peaks, notably in q1 and q6, which increases its dispersion. In contrast, Qwen is consistently the slowest in all queries, remaining in the tens of milliseconds range and reaching the worst case in q8, indicating high overhead in the end-to-end path of vector search. Complementarily, when

[8] Huggingface [12] PostgreSQL

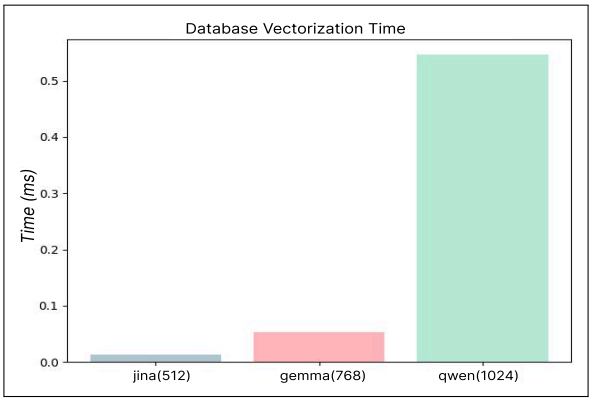

Figure 2. Comparison of tuple vectorization time among the models evaluated.

aggregating the results, the relational baseline achieved the lowest mean time (0.378799 ms), followed by the Gemma model (0.520957 ms) and Jina (1.720504 ms). In contrast, Qwen presented the highest mean time (43.394074 ms), as observed per query.

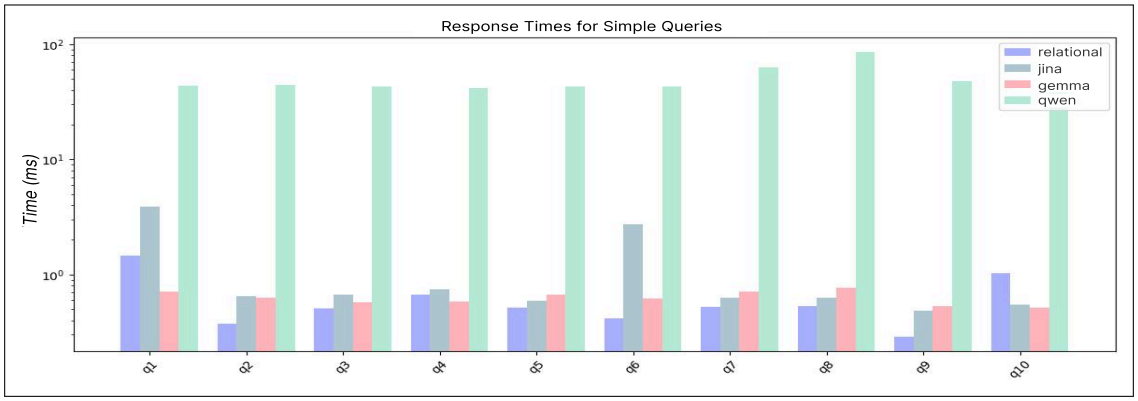

Figure 3. Pilot Study: Response Time – Simple Queries

Analogously, for the complex queries (Figure 4), the results begin to highlight the variation by query among the models. It is observed that the relational baseline and Gemma remain, in most queries, close to $\sim$1 ms, with moderate oscillations and relatively stable performance throughout the executions. Jina, in turn, presents greater dispersion and occasional latency peaks, standing out especially in q1 and, more markedly, in q6 and q10, which indicates greater sensitivity to certain query structures in the vector path. In contrast, Qwen maintains latencies consistently in the order of tens of milliseconds across all queries, with the worst case observed in q4, reinforcing the high end-to-end overhead. Complementarily, when aggregating the results, the relational baseline maintained the lowest mean time (0.972465 ms), while Gemma (1.108172 ms) and Jina (3.545881 ms) showed a moderate increase. Once again, Qwen was the model with the highest latency (46.201129 ms), suggesting higher computational cost in the pilot vector scenario.

The Gemma model was chosen because it presented the best balance between cost and performance in the pilot study: it maintained a low vectorization time (0.052503 ms) and, most importantly, achieved the lowest query latencies among the vector models, both for simple queries (0.520957 ms) and complex queries (1.108172 ms). In contrast, Qwen was discarded due to its high overhead in vectorization and querying, and Jina, although faster in vectorization, presented query times higher than those of Gemma.

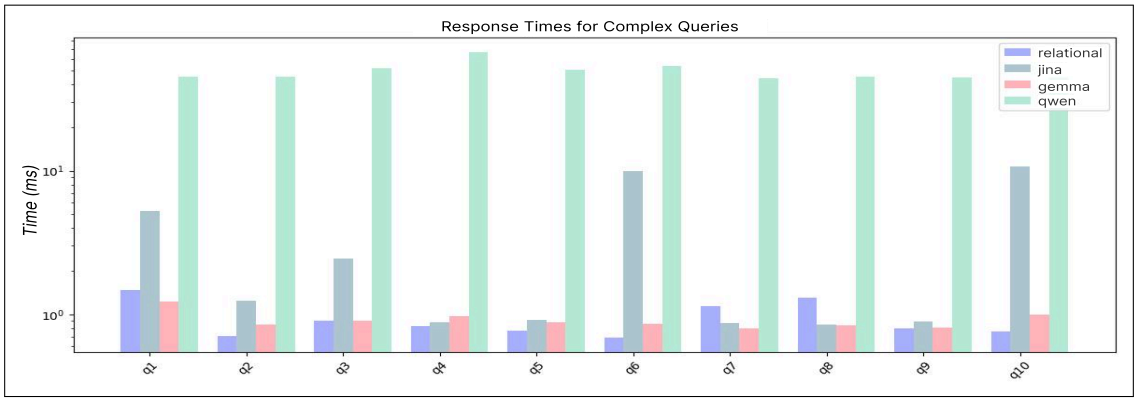

Figure 4. Pilot Study: Response Time – Complex Queries

3.2. Execution on TPC-H

For the application and testing of the architecture, the TPC-H benchmark was used with *Scale Factor* $SF = 1$, generating approximately 1 GB of synthetic data distributed across 8 tables and their 22 standard queries. The database was loaded into a PostgreSQL instance and kept unchanged throughout all measurements. After each metric collection iteration, in both execution modes, vector and relational, the query plan and the system buffers were cleared in order to reduce the experimental bias resulting from the reuse of cached data across successive executions.

3.2.1. Vectorization of tables

Figure 5 summarizes the flow of the database vectorization stage. Initially, the tuples of each relational table T are extracted in batches, in order to allow incremental and scalable processing of the database. Next, each tuple $t \in T$ is converted into a deterministic textual representation through the serialization $s(t)$ adopted in the pilot study. From this representation, vector generation is performed by computing $e(t) = \mathit{Embed}_{\mathit{Gemma}}(s(t))$ with the model `google/embeddinggemma-300m`; finally, the generated vectors are stored in the Vector Storage stage, in which, for each relational table T, a vector table $T_{\mathit{gemma\_embeddings}}$ is materialized containing the original primary key and the vector in the `vector(d)` format, with fixed $d = 768$.

Figure 5. Relational database vectorization stage, comprising the incremental extraction of tuples, their conversion into textual representation, the generation of *embeddings* with the Gemma model, and the storage of vectors in vector tables associated with the original primary keys.

3.2.2. Execution of queries

Relational execution considered the 22 TPC-H queries, executed on the database. Each query was repeated R times ($R = 20$), and the median and the 95th percentile (P95) were reported, as they respectively represent the typical behavior and the tail of the latency distribution, in addition to reducing the impact of system variations and outliers.

Vector execution used *embedding* tables derived from the TPC-H tables. For each table T, a table T_{emb} was materialized with the primary key and a d-dimensional vector. *QMAP* maps each query Q_i to a textual description $\phi(Q_i)$ and to the relevant tables $\mathcal{T}(Q_i)$, from which the query vector $v(Q_i) = \textit{Embed}(\phi(Q_i))$ was computed. The cutoff k was defined under a controlled oracle setting: a simplified relational version of each query, preserving only filtering predicates, was executed, and its resulting cardinality was used as k. Thus, k is not automatically estimated by SABRE, but experimentally defined to evaluate vector retrieval under a cutoff aligned with the relational reference set. For each $T_{emb} \in \mathcal{T}(Q_i)$, a *top-k* search was executed, returning only the configured primary key columns, as follows:

`SELECT pk[`i`] FROM` T_{emb} `ORDER BY embedding <=> (%s::vector) LIMIT %s.`

The vector $v(Q_i)$ was passed as a parameter and converted to the `vector` type in SQL. The search time is then measured per table and recorded in CSV together with i, T_{emb}, k, and the inference time of $\phi(Q_i)$. The vector version measures *top-k* retrieval per table and does not reconstruct the complete SQL result via *joins*. Fidelity is estimated by overlap with sets derived from the relational ground truth, described in Section 3.2.4.

3.2.3. Conversion and vectorization of queries

Each TPC-H query Q_i is associated with a short and deterministic textual description, $\phi(Q_i)$, designed to represent the semantic intent of the SQL query. This description is constructed based on the official specifications of the benchmark, made available on its website[13], without literal reproduction of the content. The construction of $\phi(Q_i)$ is based on the extraction of a minimal semantic signature:

1. **Target**: central entities inferred from `FROM`/`JOIN`;
2. **Metric**: aggregations and analytical functions in `SELECT`;
3. **Defining filters**: predicates in `WHERE`/`HAVING` (time, region/nation, segment, type/brand/size, thresholds);
4. **Granularity**: aggregation dimensions in `GROUP BY`;
5. **Ordering/cutoff**: `ORDER BY` and `LIMIT`;
6. **Existence/extremes**: `EXISTS`/`NOT EXISTS` and minimum/maximum.

Consider TPC-H query Q6, whose essence consists of: selecting line items (`lineitem`); applying a temporal window over `l_shipdate` (year 1994); filtering `l_discount` around 0.06 and `l_quantity` < 24; and aggregating discount revenue through `SUM`, without involving joins or complex intermediate orderings:

SQL Query example

```sql
select
  sum(l_extendedprice * l_discount) as revenue
from
  lineitem
where
  l_shipdate >= date '1994-01-01'
  and l_shipdate < date '1994-01-01' + interval '1' year
  and l_discount between .06 - 0.01 and .06 + 0.01
   and l_quantity < 24;
```

[13] https://www.tpc.org/tpc_documents_current_versions/current_specifications5.asp

The resulting semantic signature is:

- **Target:** shipped line items (`lineitem`).
- **Metric:** revenue associated with the discount (`sum(l_extendedprice * l_discount)`).
- **Filters:** shipments in 1994; discount $\approx 6\%$; quantity < 24.
- **Granularity:** not applicable (absence of `GROUP BY`).
- **Ordering/cutoff:** not applicable (absence of `ORDER BY`/`LIMIT`).

The approximate equivalent description is: *"Calculate the discount revenue of items shipped in 1994 with a discount around 6% and quantity less than 24."*

3.2.4. Metrics

To evaluate the proposed architecture, complementary quantitative and experimental metrics were considered, focused both on retrieval quality and on the execution cost of the two analyzed modes. Taken together, these metrics make it possible to verify whether the vector approach preserves the relevant elements of the relational response and what cost is associated with this retrieval when compared to traditional SQL execution.

- ***Recall@K*:** proportion of tuples from the relational output that were recovered among the results returned by the vector search. Let R_i be the set of tuples returned by the relational execution of query Q_i, and let V_i^K be the set of tuples retrieved by the vector mode in the first K positions. The metric is defined as:

$$Recall@K = \frac{|R_i \cap V_i^K|}{|R_i|}.$$

  This metric measures the coverage of the relational answer by the vector approach, indicating whether the vector search preserves the tuples considered relevant by the SQL execution.
- ***Precision@K*:** proportion of tuples retrieved in the *top-k* vector output that also belong to the relational output. Considering R_i as the relational reference set and V_i^K as the vector result set limited to K tuples, the metric is defined as:

$$Precision@K = \frac{|R_i \cap V_i^K|}{|V_i^K|}.$$

  This metric evaluates the amount of relevant information among the candidates returned by the vector search, making it possible to identify the presence of irrelevant tuples in the retrieved set.
- ***NDCG@K*:** ranking quality of the vector output considering the position of relevant tuples in the *top-k* list. Each tuple retrieved by the vector search receives a relevance value y_j, where $y_j = 1$ if the tuple belongs to the relational output and $y_j = 0$ otherwise. The metric is computed as:

$$NDCG@K = \frac{DCG@K}{IDCG@K} \quad (1) \qquad DCG@K = \sum_{j=1}^{K} \frac{2^{y_j} - 1}{\log_2(j+1)} \quad (2)$$

  This metric evaluates not only whether the vector search retrieved relevant tuples, but also whether these tuples appear in the highest positions of the ranking.
- **Fidelity:** degree of correspondence between the vector output and the relational output, adopted as *ground truth*. For its evaluation, each TPC-H query was rewritten so as to preserve only the selection predicates responsible for determining the

tuples of the final response, removing aggregations, orderings, and other operations that do not alter the set of records. Then, the primary keys of the tuples returned in the relational and vector modes were extracted. Fidelity was then measured by the overlap between these two sets.

- ***End-to-end* cost (vector latency):** total time of the vector mode for each query Q_i, calculated by the sum of the time to generate the vector $v_M(Q_i)$, from the textual description $\phi(Q_i)$, and the *top-k* search time in the vectorized tables via the cosine distance operator (<=>) of *pgvector*.
- **SQL latency:** execution time of the relational query in PostgreSQL over the original database, obtained through the `EXPLAIN ANALYZE` command and extracted from the `Execution Time` field.

4. Results

In the comparison between the outputs, vector search showed consistent performance in retrieving relevant tuples. The *Recall@K* was 0.8459, indicating that approximately 84.59% of the expected tuples were retrieved. The *Precision@K* was 0.8039, showing that most of the retrieved candidates are relevant, although some noise is still present. In turn, the *NDCG@K* reached 0.8886, indicating good ranking quality produced by vector search.

Figure 6 shows that the end-to-end latency of vector search was predominantly determined by the database query. The total cost per query presented a mean of 18,710.07 ms and a median of 20,287.31 ms, of which *embedding* accounted for 921.11 ms and 881.59 ms, in mean and median, while the database search accounted for 17,788.95 ms and 19,198.03 ms. In median terms, this corresponds to 4.69% of the cost in *embedding* and 95.31% in the query. This pattern was recurrent: in 17 of the 22 queries, the search concentrated more than 90% of the total cost and, in 12, more than 95%. In the aggregate, the *embeddings* consumed 20,264.49 ms, or 4.92% of the accumulated total of 411,621.47 ms, while the database query accounted for 95.08%, which indicates that the main cost factor of the vector approach lies in retrieval in the database.

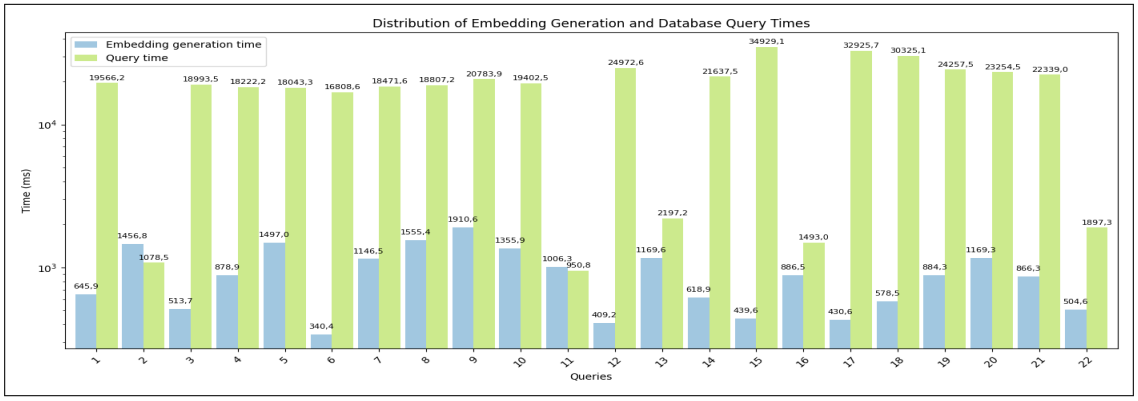

Figure 6. Vector Query Latency: Embedding and Search

Figure 7 shows the predominance of the relational mode in latency: in 21 of the 22 queries, it presented lower median and P95 times. The typical median was 385.4 ms in the relational mode and 20,287.3 ms in the vector mode, a difference of 41.6×; in P95, the typical ratio was 37×, with 525.1 ms and 22,160.8 ms, respectively. The only inversion

occurred in Q21, in which the vector mode outperformed the relational mode, with a gain of 7.9× in the median (23,205.3 ms vs 183,544.7 ms) and 9.9× in P95 (26,330.1 ms vs 260,665.2 ms). The greatest disadvantages of the vector mode occurred in Q19, Q14, Q4, and Q20, with differences between 152× and 645×. Although slower in absolute terms, the vector mode presented greater tail stability, with a median growth of 11.6% from the median to P95, compared to 33.8% in the relational mode.

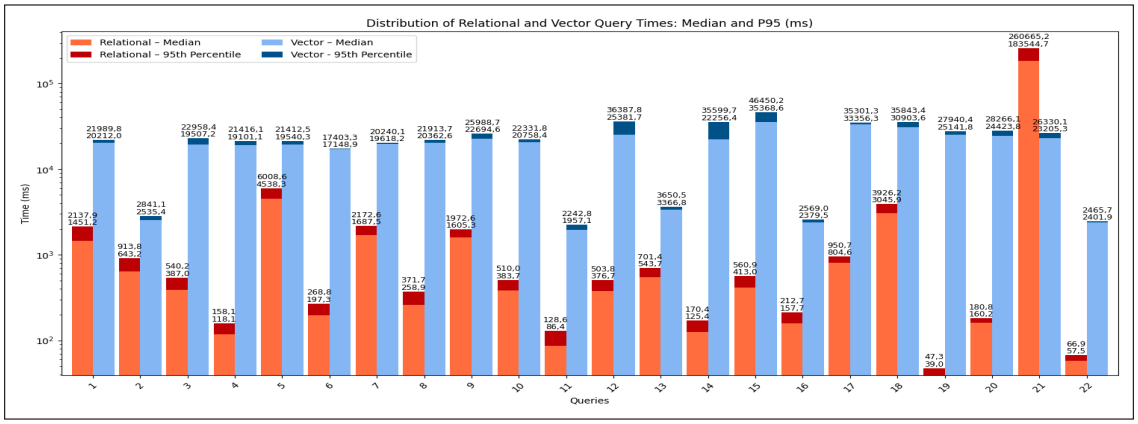

Figure 7. Median Query Time: Relational vs. Vector

4.1. Discussion

The observed metrics make it possible to analyze the architecture in terms of fidelity, cost composition, and comparative performance. The *Recall@K* of 84.59% indicates high, but not complete, coverage of the relational response; the *Precision@K* of 80.39% shows that the retrieved set is mostly relevant, although it still contains noise; and the *NDCG@K* of 88.86% indicates that relevant tuples tend to occupy higher positions in the ranking. Thus, the results do not indicate equivalence between vector search and SQL execution, but show that the architecture is capable of producing candidates semantically aligned with the expected response, characterizing it as a functional mechanism for semantic retrieval and *candidate generation*, with the need for subsequent relational refinement to ensure exactness. Regarding the *end-to-end* cost, the main bottleneck of the vector approach was not the inference of the query vector, but the database search stage. The generation of the *embedding* presented a mean of 921.11 ms and a median of 881.59 ms, whereas the vector query in the database concentrated a mean of 17,788.95 ms and a median of 19,198.03 ms, accounting for 95.08% of the accumulated cost and more than 90% of the cost in 17 of the 22 queries. Therefore, in the experimental environment, the dominant cost of the vector architecture lies in retrieval over the vector space, rather than in the processing of the input text. In the latency comparison, the relational mode showed superior performance in 21 of the 22 queries, with differences of 41.6× in the median and 37.0× in P95. This behavior is expected in a *workload* such as TPC-H, composed of complex analytical queries dependent on *joins*, aggregations, and deterministic filters, operations for which relational DBMSs are optimized. Nevertheless, in Q21, the vector mode outperformed the relational mode by 7.9× in the median and 9.9× in P95, indicating that the vector alternative can be competitive when the relational plan presents significant degradation. In addition, the vector mode showed lower dispersion, with a median growth of 11.6% from the median to P95, compared to 33.8% in the relational mode,

suggesting more predictable temporal behavior. Overall, the main practical contribution of the architecture does not lie in competing uniformly with SQL in exact queries, but in offering a semantic retrieval layer with high coverage, lexical flexibility, and potential for integration into hybrid query *pipelines*.

5. Conclusion

This work proposes an architecture for plain-text search in relational databases. The approximate coverage of the relational response by the retrieved vector set indicates that the approach preserves relevant items from the exact query, supporting its use as an initial retrieval stage. From a performance perspective, however, the evaluated vector approach incurs significantly higher cost than the relational one across most of the *workload*, with *end-to-end* latency dominated by database search rather than *embedding* generation. Consequently, the results do not support replacing SQL with vector search in complex analytical queries such as TPC-H. Instead, the vector architecture is better positioned as a complementary mechanism, particularly for queries expressed in natural language, scenarios with limited schema knowledge, tolerance to lexical variation, or approximate retrieval for exploratory analysis. In such cases, its most suitable role is as a *top-k candidate retrieval* stage, followed by relational refinement via primary keys, *join*-based recomposition, and exact predicate evaluation in SQL.

The results also delineate the main challenges for advancing the approach. As vector retrieval in the database constitutes the dominant cost, future improvements should prioritize indexing structures and execution strategies rather than focusing solely on the *embedding* model. Moreover, the evaluation is sensitive to methodological choices that directly affect system behavior, including tuple serialization $s(\cdot)$, query textualization $\phi(Q_i)$, the *embedding* model, distance metric, *top-k* selection, and DBMS operational conditions such as *cache*, parallelism, and internal maintenance. A structural limitation must also be noted: while the relational mode produces an exact final result, the vector mode yields candidate sets per table, later assessed via coverage. Thus, the comparison demonstrates semantic feasibility and relative retrieval cost, but does not constitute a one-to-one replacement of the full relational execution plan. As future work, four directions emerge: incorporating an explicit relational refinement stage after vector retrieval; performing controlled ablations on serialization and query formulation; evaluating alternative models, metrics, and ANN indexing methods; and strengthening the experimental protocol through stricter control of warm-up, resource isolation, and environmental variables. With these advances, the architecture may evolve from a semantic retrieval mechanism into an effective hybrid component for querying relational databases.

6. Acknowledgments

We gratefully acknowledge the Kunumi Institute for funding this research and for providing the computational infrastructure and support for the presentation of this work.

7. Statement on the Use of Generative AI

We do not anticipate any immediate social or ethical concerns arising from this work. Additionally, we acknowledge the use of LLMs to assist with grammar checking in parts of the manuscript.

References

Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D., Rasin, A., and Silberschatz, A. (2009). Hadoopdb: An architectural hybrid of mapreduce and dbms technologies for analytical workloads. *Proc. VLDB Endow.*, 2:922–933.

Bordawekar, R. and Shmueli, O. (2017). Using word embedding to enable semantic queries in relational databases. In *Proceedings of the 1st Workshop on Data Management for End-to-End Machine Learning*, DEEM'17, New York, NY, USA. Association for Computing Machinery.

Chen, C., Jin, C., Zhang, Y., Podolsky, S., Wu, C., Wang, S.-P., Hanson, E., Sun, Z., Walzer, R., and Wang, J. (2024a). Singlestore-v: An integrated vector database system in singlestore. *Proceedings of the VLDB Endowment*, 17(12).

Chen, J., Xiao, S., Zhang, P., Luo, K., Lian, D., and Liu, Z. (2024b). M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In *Findings of the association for computational linguistics: ACL 2024*, pages 2318–2335.

Church, K. W. (2017). Word2vec. *Natural Language Engineering*, 23(1):155–162.

Corsi, A. L. M. (2025). Estudo de caso de um sistema de gerenciamento de banco de dados vetorial para manipulação de dados de redes sociais online.

de Mendonça, A. L. C., Barioni, M. C. N., and Razente, H. (2024). Consultas analíticas por similaridade em sgbd relacionais. In *Brazilian e-Science Workshop (BreSci)*, pages 48–55. SBC.

Douze, M., Guzhva, A., Deng, C., Johnson, J., Szilvasy, G., Mazaré, P.-E., Lomeli, M., Hosseini, L., and Jégou, H. (2025). The faiss library. *IEEE Transactions on Big Data*.

Freitag, M., Bandle, M., Schmidt, T., Kemper, A., and Neumann, T. (2020). Adopting worst-case optimal joins in relational database systems. *Proceedings of the VLDB Endowment*, 13(12):1891–1904.

Garg, N., Schiebinger, L., Jurafsky, D., and Zou, J. (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, 115(16):E3635–E3644.

Gollapudi, S., Karia, N., Sivashankar, V., Krishnaswamy, R., Begwani, N., Raz, S., Lin, Y., Zhang, Y., Mahapatro, N., Srinivasan, P., et al. (2023). Filtered-diskann: Graph algorithms for approximate nearest neighbor search with filters. In *Proceedings of the ACM Web Conference 2023*, pages 3406–3416.

Greiner-Petter, A., Youssef, A., Ruas, T., Miller, B. R., Schubotz, M., Aizawa, A., and Gipp, B. (2020). Math-word embedding in math search and semantic extraction. *Scientometrics*, 125(3):3017–3046.

He, D., Nakandala, S., Banda, D., Sen, R., Saur, K., Park, K., Curino, C., Camacho-Rodríguez, J., Karanasos, K., and Interlandi, M. Query processing on tensor computation runtimes.

Khalifeh, F., Taheri, M., Mansoori, E., and Fakhrahmad, M. (2025). Enhancing keyword search in relational databases with word embeddings. *IEEE Access*.

Kumar, P. (2024). Large language models (llms): survey, technical frameworks, and future challenges. *Artificial Intelligence Review*, 57(10):260.

Liu, J., Zhang, Y., Jin, C., Gupta, A., Liu, S., and Wang, J. (2026). Fast vector search in postgresql: A decoupled approach. In *Conference on Innovative Data Systems Research (CIDR)*.

Mama, R. and Machkour, M. (2025). Semantic and fuzzy integration: A new approach to efficient and flexible querying of relational databases. *International Journal of Advanced Computer Science & Applications*, 16(5).

Mathur, S. and Chhabra, A. (2024). Vector search algorithms: A brief survey. In *2024 4th International Conference on Ubiquitous Computing and Intelligent Information Systems (ICUIS)*, pages 365–371. IEEE.

Pan, J. J., Wang, J., and Li, G. (2024a). Survey of vector database management systems. *The VLDB Journal*, 33(5):1591–1615.

Pan, J. J., Wang, J., and Li, G. (2024b). Vector database management techniques and systems. In *Companion of the 2024 International Conference on Management of Data*, pages 597–604.

Petrola, L., Brayner, A., and Franco, W. (2025). Heuristic-guided text-to-sql translation with llms: Optimizing natural language interfaces for relational databases. In *Simpósio Brasileiro de Banco de Dados (SBBD)*, pages 126–139. SBC.

Riccardo Cappuzzo, P. P. e. S. T. (2020). Criação de incorporações de conjuntos de dados relacionais heterogêneos para tarefas de integração de dados. *Anais da Conferência Internacional ACM SIGMOD de 2020 sobre Gerenciamento de Dados*.

Wei, C., Wu, B., Wang, S., Lou, R., Zhan, C., Li, F., and Cai, Y. (2020). Analyticdb-v: A hybrid analytical engine towards query fusion for structured and unstructured data. *Proc. VLDB Endow.*, 13(12):3152–3165.

Yong, K. K. and Teh, P. C. (2025). Evaluating gpu-accelerated structured query engines using tpc-h: A comparative study with duckdb and rapids. In *2025 IEEE 11th International Conference on Smart Instrumentation, Measurement and Applications (ICSIMA)*, pages 144–149. IEEE.